LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-594552

Performance Characterization and Validation of mocfe_bone

A. Bhatele, M. Schulz

October 23, 2012

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Performance Characterization and Validation of `mocfe_bone` [†]

Contribution to the CESAR Annual Report FY12

Abhinav Bhatele and Martin Schulz

Center for Applied Scientific Computing, Lawrence Livermore National Laboratory
E-mail: {bhatele, schulzm}@llnl.gov

1 Introduction

In the first phase of CESAR, the performance analysis task focused on understanding the baseline performance characteristics of `mocfe_bone` — studying its behavior under different parallelization scenarios and comparing its scaling behavior to the MOCFE solver in the main application, UNIC. The results will help drive the co-design process for this application, which is one of the three target applications in CESAR, and will help identify current and future scaling bottlenecks.

2 Degrees of Parallelism in `mocfe_bone`

There are three different dimensions along which MOCFE (and `mocfe_bone`) can be parallelized: space, angle and energy. Accordingly, `mocfe_bone` takes as input the size of the mesh, the number of angles and the number of energy groups. Table 1 lists the arguments to the proxy-app and their definitions.

Argument	Detailed Explanation
`MeshScale`	size of mesh in one dimension (on each MPI process)
`ParallelInX`	number of processes in X
`ParallelInY`	number of processes in Y
`ParallelInZ`	number of processes in Z
`Angle_Visible`	total number of angles
`AnglesPerProcess`	number of angles per MPI process
`Group_Visible`	total number of energy groups
`GroupsPerProcess`	number of energy groups per MPI process

Table 1: Command line arguments to `mocfe_bone`

The mesh scale and the number of processors used in each dimension together define the underlying physical domain and its size. Additionally, the number of angles defines the number of directions from which the trajectories intersect the physical domain. Energy groups, are loosely coupled and can be parallelized across processors.

[†]This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-TR-594552).

3 Sequential Performance

As part of the co-design effort, we have been running `mocfe_bone` on Blue Gene/P (Challenger/ Intrepid) and Blue Gene/Q (Vesta) at ANL and a Xeon-Infiniband cluster (Cab) at LLNL. We first looked at the instruction mix for the proxy-app when doing parallelization in angles, energy, space or a combination of these. Table 2 presents the number of branches, loads, stores, cache misses and flops per cycle for different configurations.

Run configuration	**1/16/1**	**1/16/8**	**32/16/1**	**32/16/8**
Branches	0.362	0.363	0.522	0.731
Loads	0.061	0.061	0.034	0.019
Stores	0.027	0.027	0.021	0.011
Cache misses	0.003	0.003	0.055	0.112
Flops	0.133	0.133	0.135	0.127

Table 2: Instruction mix for sequential runs on one node of Blue Gene/P (SMP mode).

The three numbers in each column of the first row (separated by /) represent the mesh size, number of angles and number of energy groups respectively. For example, the first column represents a run with a mesh of 1^3, 16 angles and 1 energy group, and the last column represents a run with a mesh of 32^3, 16 angles and 8 energy groups. It is evident that the number of branches per cycle is high for all configurations and increases with the number of energy groups and mesh size. The number of flops per cycle does not change irrespective of the mesh size, however, the number of cache misses increases drastically for a mesh of 32^3 instead of 1^3.

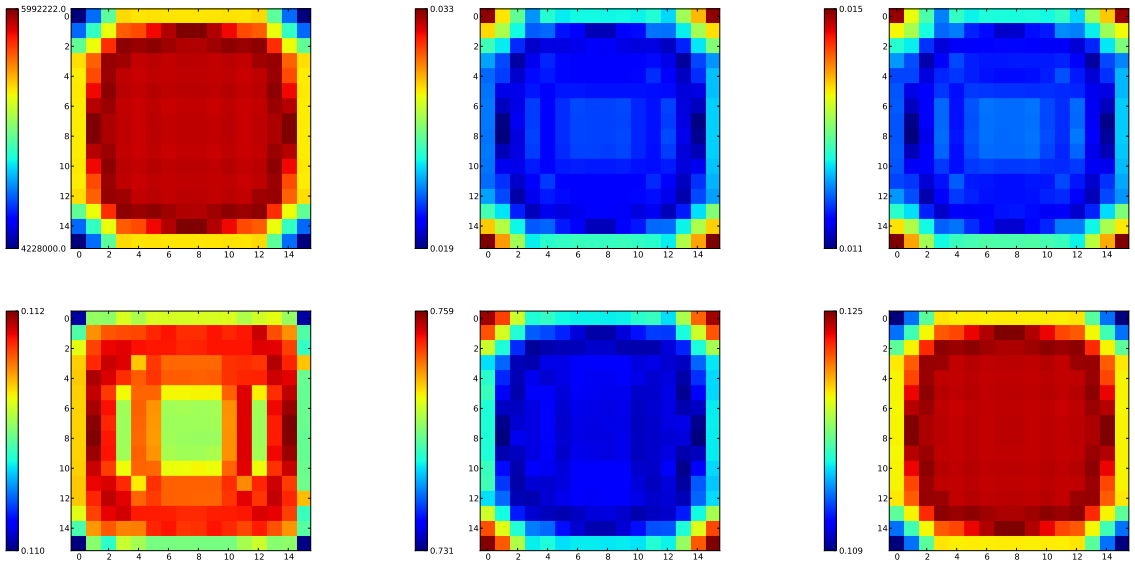

Figure 1: From top left to bottom right: number of interactions, hardware counter data showing loads, stores, cache misses, branches and total flops per cycle mapped to the 16×16 mesh topology.

4 Performance Analysis through Visualizations

We studied the performance of `mocfe_bone` with a parallel domain decomposition in two dimensions instead of three on 256 processors (by keeping the Z dimension of processors, i.e., the `ParallelInZ` parameter at 1). We used a mesh size of 32^3 on each processor with a parallel decomposition of MPI processes of $16 \times 16 \times 1$. The number of angles and energy groups was 16 and 8 respectively.

The top left graphic in Figure 1 shows the application space and the associated workload in terms of the number of interactions mapped to the underlying 16×16 mesh. Other code characteristics (like number of trajectories) show an almost identical distribution. To study the correlation of hardware counter values for loads, stores, branches, cache misses and floating point operations (as we presented them above for the sequential case), we map the values obtained on each node to the same domain (see rest of Figure 1).

We can clearly see that the structure of the problem directly matches the performance characteristics. While the number of floating point operations directly correlates with the work in the simulation domain, loads, stores and branches are inversely related. The cache miss pattern is more involved and needs further exploration. The problem boundaries require more branches, loads, and stores and less flops.

5 Scaling Properties

We further studied the strong scaling performance of `mocfe_bone` to validate it's performance against that of MOCFE. We used a mesh of 32^3 on one node and did strong scaling in space (keeping the global mesh size the same and reducing the mesh size on each processor by half as we doubled the number of processors). The number of angles and energy groups was fixed at 16 and 8 respectively and the trajectory spacing was $0.01cm^2$ to match the existing MOCFE data.

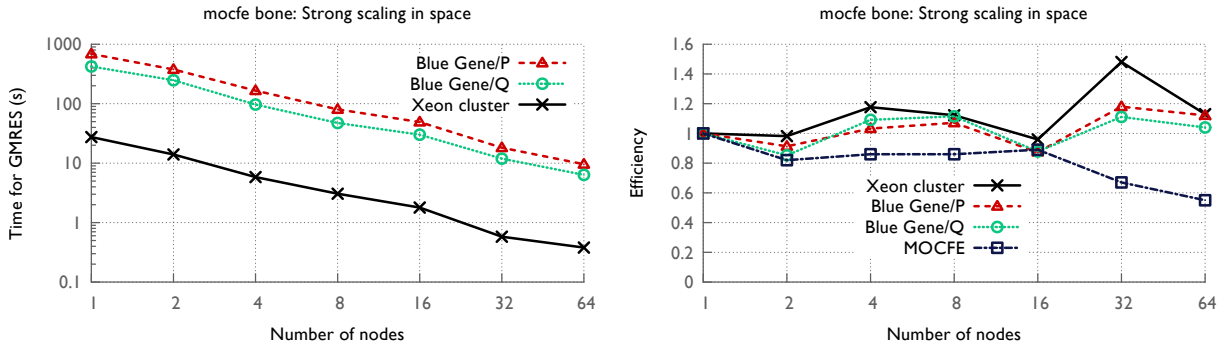

Figure 2: Strong scaling efficiency of `mocfe_bone` compared to MOCFE

Figure 2 (left) shows the strong scaling performance of `mocfe_bone` running from 1 to 64 nodes on Blue Gene/P, Blue Gene/Q and Cab. Figure 2 (right) compares its efficiency with that of MOCFE. We see a similar scaling behavior on all three platforms. Compared to MOCFE, we see a reasonable match for lower node counts, but for larger node counts trends seem to diverge (based on the data from MOCFE that we currently have). These results are not surprising, since the fundamental structure of MOCFE and `mocfe_bone` is different: while the computation being done is very similar, the codes perform them on different domains. MOCFE is an unstructured code, while `mocfe_bone` is structured. We therefore will need a second proxy application for this code that can mimic this behavior and provide input on how to optimize scaling.